Question 2

In Processing using Python mode, you aim to create an application that displays an image named "scenery.jpg" in full screen. The canvas is set to full screen mode and is known to have the same aspect ratio as the image. How would you implement the code to load and display the image so that it covers the entire canvas without distorting its proportions?

*def setup():*

*fullScreen()*

*global img*

*img = loadImage("scenery.jpg")*

*imageMode(CENTER)*

*def draw():*

*background(0)*

*image(img, width / 2, height / 2, width, height)*

3. In Processing using Python mode, you're designing a basic slideshow program to display three different images sequentially on the canvas. The images are named "image1.jpg", "image2.jpg", and "image3.jpg". The slideshow should start with "image1.jpg", and each time a key is pressed, it should advance to the next image. After "image3.jpg" is displayed, the next key press should return the slideshow to "image1.jpg". How would you implement a state machine using if statements and keyboard input to cycle through the images?

*currentImage = 1*

*def setup():*

*global img1, img2, img3*

*size(800, 600)*

*img1 = loadImage("image1.jpg")*

*img2 = loadImage("image2.jpg")*

*img3 = loadImage("image3.jpg")*

*imageMode(CENTER)*

*def draw():*

*global currentImage*

*if currentImage == 1:*

*image(img1, width / 2, height / 2)*

*elif currentImage == 2:*

*image(img2, width / 2, height / 2)*

*elif currentImage == 3:*

*image(img3, width / 2, height / 2)*

*def keyPressed():*

*global currentImage*

*currentImage += 1*

*if currentImage > 3:*

*currentImage = 1*

Question 4

In Processing using Python mode, you are given the task of reading data from a CSV file named "Data.csv" that consists of a header line followed by several rows of data, each with two columns. The first column contains names, and the second column contains corresponding ages. You need to read and process this data, skipping the header line. Considering the requirement to use createReader() for file reading, which of the following code snippets correctly implements the reading process, starting with the skipping of the header line?

*def setup():*

*global reader*

*reader = createReader("Data.csv")*

*reader.readLine()*

*line = reader.readLine()*

*while line != None:*

*data = line.split(",")*

*print("Name:", data[0], "Age:", data[1].strip())*

*line = reader.readLine()*

Question 5

In Processing using Python mode, you are tasked with creating a visualization of Perlin noise on the screen. The visualization should exclusively display two colors: green for areas corresponding to the top 20% of the Perlin noise values (indicating higher elevations or intensities) and black for the rest. How would you implement this to ensure that only the highest 20% of Perlin noise values are shown in green, with the remaining area in black?

*def draw():*

*loadPixels()*

*for x in range(width):*

*for y in range(height):*

*noiseVal = noise(x \* 0.01, y \* 0.01)*

*if noiseVal > 0.8:*

*pixels[y \* width + x] = color(0, 255, 0)*

*else:*

*pixels[y \* width + x] = color(0)*

*updatePixels()*

Question 6

In Processing using Python mode, you are working with an image loaded into your sketch. Your task is to create a new visualization layer where only the top 10% brightest pixels of the original image are displayed in white, and all other pixels are shown in black. This involves evaluating the brightness of each pixel in the image and applying the specified threshold. Which of the following code snippets correctly implements this effect, ensuring that only the brightest pixels are highlighted in white?

*def setup():*

*global img*

*size(600, 400)*

*img = loadImage("picture.jpg")*

*img.loadPixels()*

*def draw():*

*loadPixels()*

*for i in range(len(img.pixels)):*

*pixelColor = img.pixels[i]*

*brightnessValue = brightness(pixelColor)*

*if brightnessValue > 225:*

*pixels[i] = color(255)*

*else:*

*pixels[i] = color(0)*

*updatePixels()*

Question 7

In Processing using Python mode, you are working with an image that initially has a resolution of 1000x1000 pixels. Your task is to re-sample this image down to a resolution of 64x64 pixels, effectively creating a smaller version of the original image. Which of the following code snippets correctly performs this re-sampling, ensuring the image is resized accurately?

*def setup():*

*global img*

*size(64, 64)*

*img = loadImage("highResImage.jpg")*

*img.resize(64, 64)*

*image(img, 0, 0)*

Question 7

In Processing using Python mode, you are working with an image that initially has a resolution of 1000x1000 pixels. Your task is to re-sample this image down to a resolution of 64x64 pixels, effectively creating a smaller version of the original image. Which of the following code snippets correctly performs this re-sampling, ensuring the image is resized accurately?

*def setup():*

*global img*

*size(64, 64)*

*img = loadImage("highResImage.jpg")*

*img.resize(64, 64)*

*image(img, 0, 0)*

Question 8

In Processing using Python mode, you are creating an interactive sketch where an image is displayed on the canvas. You want to implement a feature that samples the color of the image at the mouse's current location and then draws a rectangle filled with that sampled color, effectively making the rectangle act as a real-time color sampler that follows the mouse. Assuming the image has already been loaded and displayed on the canvas, which of the following code snippets correctly implements this color sampling feature?

*def setup():*

*global img*

*size(800, 600)*

*img = loadImage("picture.jpg")*

*image(img, 0, 0)*

*def draw():*

*loadPixels()*

*x = mouseX*

*y = mouseY*

*index = y \* width + x*

*colorSample = pixels[index]*

*fill(colorSample)*

*rect(mouseX, mouseY, 50, 50)*